

# Rapport TP 3 Mobile

Franck Reveille

Mars 2025

## 1 Introduction

Ce rapport porte sur le troisième TP de l'UE de programmation mobile. Voici mon [dépôt github contenant l'application](#)

L'application a été testée sur mon appareil personnel, à savoir un Samsung Galaxy S23. Comme au précédent TP, l'application utilise Material. Voici la [documentation](#) et le [catalogue](#) de Material.

Contrairement aux précédents TPs, celui-ci est en Kotlin et non pas en Java ce qui a imposé plus d'apprentissages.

Pour la persistance, j'ai opté pour l'utilisation de l'ORM ROOM pour sa modernité et l'intérêt d'apprendre un ORM un peu particulier car sous Android.

Voici l'architecture des fichiers Java de l'application :

```
com.example.persistence
├── activities
│   ├── FirstActivity
│   └── PlanningActivity
├── dao
│   ├── PlanningDao
│   └── UserDao
├── entities
│   ├── DailyPlanning
│   └── User
├── fragments
│   ├── ConfirmFragment
│   ├── LoginFragment
│   ├── NewPlanningFragment
│   ├── SeePlanningFragment
│   └── SignUpFragment
├── viewModels
│   ├── FirstViewModel
│   ├── LoginViewModel
│   ├── PlanningViewModel
│   └── SignUpViewModel
├── AppDataBase.kt
└── Converters
```

FIGURE 1 – Architecture

- Converters est une classe utilitaire servant à convertir certains objet, particulièrement les objets LocalDate (non supporté par Room) en Long.

## 2 Images des différentes vues

21:34 5G 31%

# Inscription

Identifiant

Mot de passe

Prenom

Nom

Date de naissance

N° de telephone

Adresse mail

Sport Musique Film Lecture

Valider

*Vous avez déjà un compte ?*

FIGURE 2 – Signup Fragment

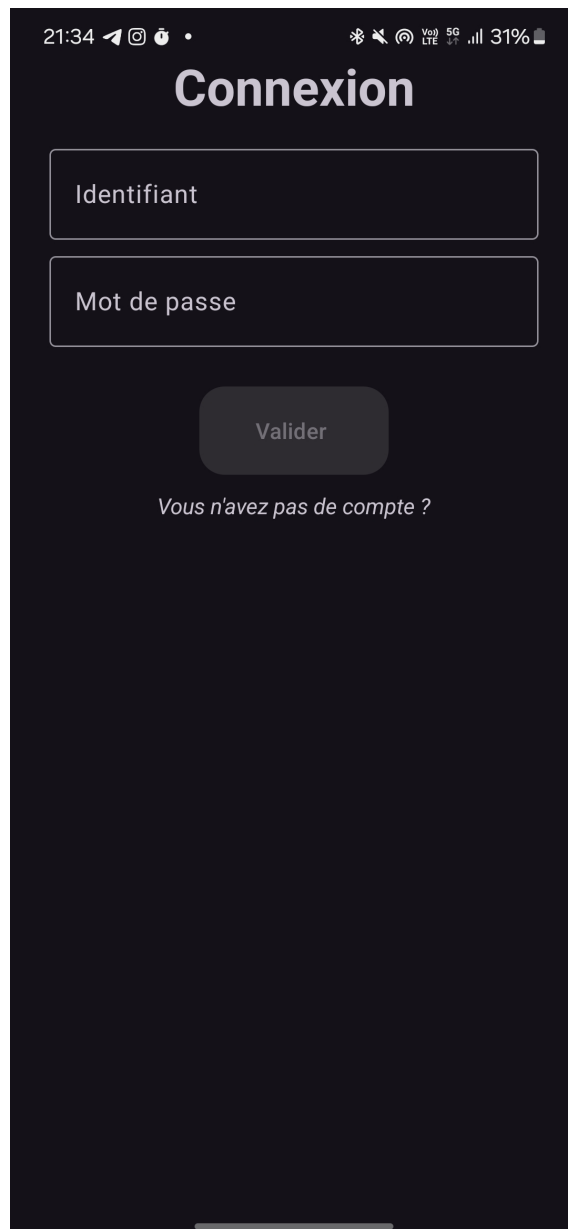
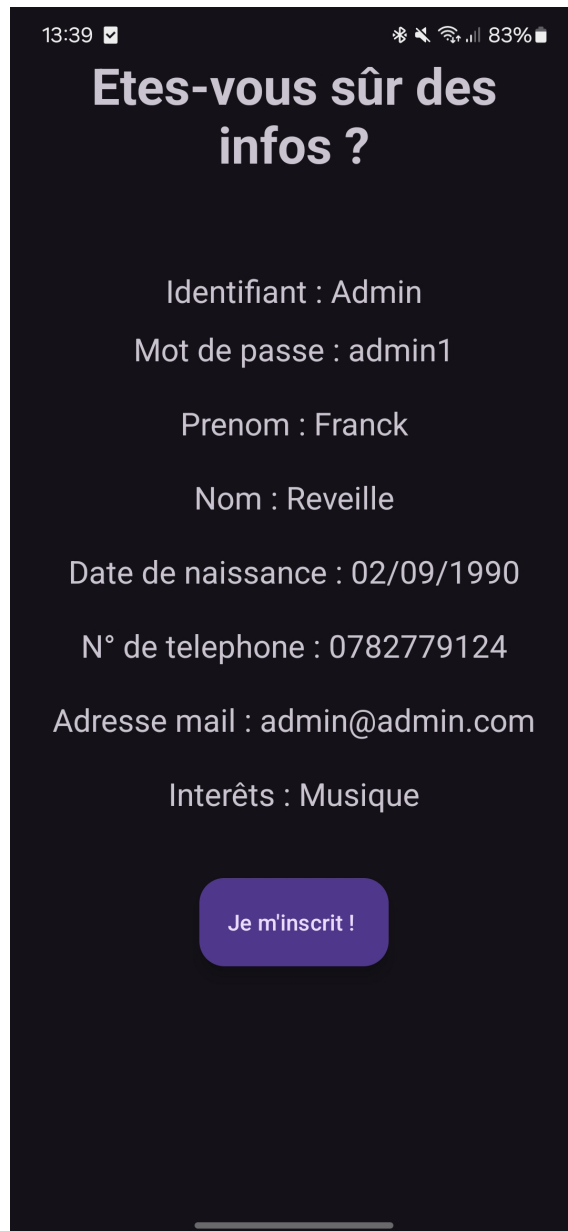


FIGURE 3 – Login Fragment

A mobile application registration screen with a dark background. At the top, the status bar shows the time 13:39, a checkmark icon, and various connectivity and battery icons (83%). The main heading is 'Etes-vous sûr des infos ?' in large white font. Below it, several fields are listed in white text: 'Identifiant : Admin', 'Mot de passe : admin1', 'Prenom : Franck', 'Nom : Reveille', 'Date de naissance : 02/09/1990', 'N° de telephone : 0782779124', 'Adresse mail : admin@admin.com', and 'Interêts : Musique'. At the bottom, there is a purple rounded button with the text 'Je m'inscrit !' in white. A thin horizontal line is visible at the very bottom of the screen.

13:39 ✓

✱ 83%

# Etes-vous sûr des infos ?

Identifiant : Admin

Mot de passe : admin1

Prenom : Franck

Nom : Reveille

Date de naissance : 02/09/1990

N° de telephone : 0782779124

Adresse mail : admin@admin.com

Interêts : Musique

Je m'inscrit !

FIGURE 4 – Enter Caption

A mobile application interface for creating a schedule. The title is "Créer votre planning". Below the title are four rectangular buttons stacked vertically, each containing a time slot: "08h-10h", "10h-12h", "12h-14h", and "14h-16h". Below these buttons is a rounded rectangular button labeled "Valider". The status bar at the top shows the time 21:35, various icons, and a battery level of 31%.

21:35 5G 31%

# Créer votre planning

08h-10h

10h-12h

12h-14h

14h-16h

Valider

FIGURE 5 – Creation Fragment



FIGURE 6 – Consultation Fragment

### 3 Explication de l'architecture

L'application est composée de deux activités, une gérant les fragments de connexion/inscription et une permettant la création/consultation du planning.

On n'a donc que deux objets à stocker en BDD, utilisateurs et planning dont voici les entités :

```
1 @Entity(tableName = "users")
2 data class User(
3     @PrimaryKey val login: String,
4     val password: String,
5     val firstName: String,
6     val lastName: String,
7     val birthDate: LocalDate,
8     val phoneNumber: String,
9     val email: String,
10    val interests: List<String>
11 )
12
13
```

```

14 @Entity(tableName = "daily_planning",
15     foreignKeys = [
16         ForeignKey(
17             entity = User::class,
18             parentColumns = ["login"],
19             childColumns = ["userId"],
20             onDelete = ForeignKey.CASCADE
21         )
22     ])
23 data class DailyPlanning(
24     @PrimaryKey(autoGenerate = true) val id: Int = 0,
25     val userId: String,
26     val date: Long,
27     val slot1Activity: String, // 08h-10h
28     val slot2Activity: String, // 10h-12h
29     val slot3Activity: String, // 14h-16h
30     val slot4Activity: String // 16h-18h
31 )

```

Chacun est donc logiquement doté d'un DAO :

```

1 @Dao
2 interface UserDao {
3     @Insert
4     suspend fun insertUser(user: User)
5
6     @Query("SELECT * FROM users WHERE login = :login")
7     suspend fun getUserByLogin(login: String): User?
8
9     @Query("SELECT COUNT(*) FROM users WHERE login = :login")
10    suspend fun checkLoginExists(login: String): Int
11
12    @Query("SELECT * FROM users WHERE login = :login AND password = :password")
13    suspend fun checkLoginInfos(login: String, password : String) : User?
14 }
15
16
17 @Dao
18 interface PlanningDao {
19
20     @Insert
21     suspend fun insertPlanning(planning: DailyPlanning)
22
23     @Query("SELECT * FROM daily_planning WHERE userId = :userId AND date = :date")
24     suspend fun getTodayPlanningForUser(userId: String, date: Long): DailyPlanning
25
26     @Query("SELECT COUNT(*) FROM daily_planning WHERE userId = :userId AND date = :date")
27     suspend fun checkPlanningExists(userId: String, date: Long): Int
28
29 }

```

Chaque méthode annotée "suspend" est asynchrone et devra être appelée dans une coroutine, mais nous verrons cela plus tard.

Ce TP intègre la notion de ViewModel et de LiveData. L'application les utilise pour gérer plusieurs données entre fragments, mais aussi l'affichage de fragments par une activité.

Les classes FirstViewModel, LoginViewModel et SignUpViewModel concernent la première activité et ses fragments tandis que PlanningViewModel concerne la seconde.

- FirstViewModel : Stocke le fragment à afficher
- LoginViewModel : Stocke simplement les informations de connexion saisies par l'utilisateur
- SignUpViewModel : Contient les saisies utilisateurs, des méthodes de vérification de regex sur chaque champs de saisie ainsi avec le message d'erreur correspondant si le regex n'est pas respecté et enfin une méthode de création d'un objet User à partir des saisies utilisateurs.

Voici un extrait du code mentionné :

```
1  ...
2  private val _firstNameError = MutableLiveData<String?>()
3      val firstNameError: LiveData<String?> = _firstNameError
4
5  private val _lastNameError = MutableLiveData<String?>()
6      val lastNameError: LiveData<String?> = _lastNameError
7
8  private val _phoneError = MutableLiveData<String?>()
9      val phoneError: LiveData<String?> = _phoneError
10
11 private val _emailError = MutableLiveData<String?>()
12     val emailError: LiveData<String?> = _emailError
13
14 fun validateLogin(login: String) {
15     val regex = Regex("[A-Za-z][A-Za-z0-9]{0,9}$")
16     _loginError.value = if (!login.matches(regex)) "Identifiant
17         incorrecte" else null
18 }
19
20 fun validatePassword(password: String) {
21     _passwordError.value = if (password.length < 6) "Mot de passe
22         trop court" else null
23 }
24
25 fun validateFirstName(firstname: String) {
26     val regex = Regex("[A-Za-z]+$")
27     _firstNameError.value = if (!firstname.matches(regex)) "
28         Caract re invalide" else null
29 }
30
31 fun toUser(): User {
32     val user = User(login.value.toString(),
33         password.value.toString(),
34         firstname.value.toString(),
35         lastname.value.toString(),
36         birthdate.value!!,
37         phone.value.toString(),
38         email.value.toString(),
39         interests.value!!)
40     return user
41 }
42 ...
```

```
1 class FirstViewModel : ViewModel() {
2     private val _currentFragment = MutableLiveData<String>()
3     val currentFragment: LiveData<String> = _currentFragment
4
5     fun setCurrentFragment(fragmentTag: String) {
6         _currentFragment.value = fragmentTag
7     }
8 }
```



### 3.1 Activités

Les activités ont besoin de savoir quand est-ce qu'un fragment communiquent avec elles. Pour se faire, les fragments définissent une interface de callback que l'activité implémentera :

Dans SignUpFragment.kt :

```
1 private var signupListener: SignupFragmentInterface? = null
2
3 interface SignupFragmentInterface{
4     fun onSwitchToLogin()
5     suspend fun signup()
6 }
7
8 override fun onAttach(context: Context) {
9     super.onAttach(context)
10    if (context is SignupFragmentInterface) {
11        signupListener = context
12    } else {
13        throw RuntimeException("$context must implement
14                                OnSwitchToLoginInterface")
15    }
16 }
17 .
18 .
19 submitButton = parent.findViewById<ExtendedFloatingActionButton>(R.id.
20    submitSignUpButton)
21    submitButton.setOnClickListener { lifecycleScope.launch {
22        Log.d("Fragment", "Signup clicked")
23        viewModel.setLogin(loginText.text.toString())
24        viewModel.setPassword(passwordText.text.toString())
25        viewModel.setFirstname(firstnameText.text.toString())
26        viewModel.setLastname(lastnameText.text.toString())
27        viewModel.setBirthdateString(birthdateInputEdit.text.
28            toString())
29        viewModel.setBirthdate(date)
30        viewModel.setPhone(phoneText.text.toString())
31        viewModel.setEmail(mailText.text.toString())
32        signupListener?.signup()
33    } }
```

Dans FirstActivity.kt :

```
1 override fun onSwitchToLogin() {
2     Log.d("switch", "CallBack")
3     viewModel.setCurrentFragment("login")
4 }
5
6 override suspend fun signup() {
7     Log.d("signup", signUpViewModel.login.value.toString())
8     val counts = userDao.checkLoginExists(signUpViewModel.login.value.
9         toString())
10    if (counts>0){
11        Snackbar.make(rootView, "Un compte est d j associ cet
12            identifiant", Snackbar.LENGTH_SHORT).show()
13    }
14    else{
15        viewModel.setCurrentFragment("confirm")
16    }
17 }
```

Certaines parties du code sont lancées via `lyfecyleScope.launch{}` ce qui permet de lancer une coroutine, nécessaire pour lancer les méthodes `suspend` en kotlin.